SECURE NETWORK USER STATES

CONTINUING DATA

5   This is a CONTINUATION of, and incorporates by reference in its entirety, United States Application Serial No. **09/545,009**, entitled SECURE INTERNET USER STATE CREATION METHOD AND SYSTEM WITH USER SUPPLIED KEY AND SEEDING and filed on April 7, 2000 by Wallace, which in turn is a CONTINUATION of, and incorporates in its entirety, United States Application Serial No. **09/475,638**

10   entitled METHOD AND SYSTEM FOR CREATING SECURE INTERNET USER STATES, filed on December 30, 1999 by Wallace et. al. Further, this disclosure is related to, and incorporates in by reference in their entireties, United States Application Serial Nos. **09/491,225** entitled KEY-BASED METHOD AND SYSTEM FOR CREATING SECURE INTERNET USER STATES, and **09/491,059** entitled KEY-

15   BASED METHOD AND SYSTEM WITH KEY INDEX FOR CREATING SECURE INTERNET USER STATES, both, filed on January 25, 2000 by Wallace et. al, and now abandoned; and United States Application Serial No. **xx/xxx,xxx** entitled KEY-BASED SECURE NETWORK USER STATES filed on October 31, 2001 by Wallace et al.

20   FIELD OF INVENTION

        The present invention relates to secure network user states.

BACKGROUND OF THE INVENTION

        Computer networks, such as the Internet, are well known in the art, and may be

25   based on the HTTP protocol. Because HTTP is a stateless, or non-persistent, protocol, it is not possible for such servers to differentiate between visits by a specific user unless the server can somehow mark the user to create a state or logical nexus between the server and the user. Thus, each visit by an Internet user to a website is unique, in that the website does not generally know the identity of the user and/or other information about

30   the user, with the exception of a few details such as browser type, IP address, etc. It should be noted, however, that when a user has a fixed IP address, the user's identity or

information about the user may be known by logical relation to a database. But, since the majority of Internet users are assigned dynamic IP addresses each time they connect to the Internet, reliance on a user's IP address to create a state is problematic since their IP addresses may change each time a user connects to the Internet.

5   To remedy the problem of HTTP's stateless nature, cookies have been introduced for the specific purpose of creating states. They may be temporary, in which case they are stored only in memory; or persistent, in which case they are stored in a file, typically on a hard drive, for period of time measured by an expiration date field of a cookie. A cookie may be thought of as a data structure stored in the memory or on the storage

10   device of a user's computer, with the cookie containing data, such as the user's identity and/or other information about the user for the purpose of creating a state between the web server and the user. Thus, when a user visits a particular website, a cookie stored on a user's computer may be sent from the user's computer over the Internet to the web server, which then extracts the data from the cookie, processes the data and therewith

15   creates a state. For example, a user's name may be stored in a cookie and when that user visits a particular website, the data contained in the cookie may be sent to the server and used to identify the user.

        More specifically and typically, when a user first visits an Internet website, a web server associated with the website may send a cookie to the user, which is then stored in

20   the memory or on the hard drive of a user's computer, in conjunction with the user's Internet browser software. When the user subsequently visits the website, the cookie may be sent back to the server so that the user's identity and/or other information about the user that is stored in the cookie may be known to the server via the data contained in the cookie, such that a state between the user and the web server is created.

25   However, the use of cookies has created a significant problem relating to user privacy. Because these cookies are stored on a user's computer, especially when on a hard drive, other servers may potentially access the cookies of other servers and extract and read the user's identity and/or other information about the user that is stored in those cookies. Such extracting and reading is considered by many as an invasion of the user's

30   privacy.

An attempted solution to protect the privacy of Internet users is provided in RFC 2109, HTTP STATE MANAGEMENT MECHANISM, having a publication date of February, 1997. This solution involves a domain restriction on reading and writing cookies, which must be implemented in conjunction with a user's particular browser software for effectuation. For example, a web server associated with the domain thissite.com may write a cookie having the domain value .thissite.com. According to the domain restriction, this cookie may only be read by a server within the specified domain and related sub-domains. For example, while the servers at thissite.com, L1.thissite.com, L2.L1.thissite.com, etc. may read the cookie having the domain value .thissite.com, the servers othersite.com, L1.othersite.com, L2.L1.othersite.com may not read the cookie having the domain value .thissite.com. While this methodology appears adequate on its face, practically it is not. It suffers from at least four deficiencies.

A first problem is that this methodology requires software vendors producing browser software to implement this domain restriction. While mainstream vendors may attempt to comply, other smaller vendors may not. Thus, failed compliance may create a hole through which a user's privacy may be invaded via the unauthorized access of cookies despite the existence of a domain restriction.

A second problem is that despite attempted compliance, one or more bugs or exploits in the browser software may exist and be exploited; thus, also creating a hole through which a user's privacy may be invaded. For example, as identified in the article, COOKIE EXPLOIT, published by COOKIE CENTRAL on December 14, 1998, such a bug did exist and a hole was potentially created and exploited. The bug allowed cookies to be shared between unrelated domains, despite the domain restriction implemented by some if not all cookie-based Internet browser applications. Basically, by concatenating an ellipsis ("...") at the end of the domain value set in a cookie, other unrelated servers were able to read those cookies. Such a domain value may be ".thissite.com..." According to this article, at the time of publication all mainstream Internet browser applications were vulnerable to this exploit. Indeed, the article goes on to assert that the most popular Internet browser applications, INTERNET EXPLORER and NETSCAPE, were known to be vulnerable on the WINDOWS, MAC and LINUX platforms. Thus, the domain restriction was nullified and servers participating in the exploitation of this bug

were able to access cookies from domains outside their own domain, which is exactly what the domain restriction of RFC 2109 was intended to prevent. Thus, the privacy of Internet users benefiting from the use of cookies was unequivocally subject to invasion.

A third problem is that the cookies stored on a user's hard drive may be viewed by a person who is physically using the user's computer. The location and naming of cookie files stored on a user's hard drive are generally known or discoverable by those skilled in the art. For example, it is well known in the art that the browser software application NETSCAPE™ that is developed and distributed by NETSCAPE COMMUNICATIONS CORPORATION™ generally stores cookies in a user directory in a single file named "cookie.txt". One physically using a user's computer may open such a file with a simple text editor and directly view and/or print the data contained in all cookies present, which is clearly an invasion of the user's privacy.

A fourth problem is that under certain conditions servers may directly read cookie files outside the domain restriction set in the cookies. It is generally known in the art that where a user's Internet browser software is configured to enable JAVA script, specific files having a known name (such as, "cookies.txt") may be directly accessed, read and transmitted to some location over the Internet by a "virus" embedded within such JAVA script. Additionally, a devious program may also contain such a virus that can do the same. Many Internet users download and run executable programs from the Internet knowingly and unknowingly risking the infection of a virus; and therefore, this risk is present and real. The location of cookie files are generally known or discoverable to those ordinarily skilled in the art. Indeed, such a virus may execute a "directory" command to obtain the names of files and directories on a hard drive; for example, a directory listing of files and directories in the "c:\windows\Temporary Internet Files" directory or "c:\Program Files\Netscape\Users" directory. The former may produce cookie files produced by INTERNET EXPLORER; while the latter may produce the names of the directories of users of NETSCAPE (i.e., John), which may be used to access the NETSCAPE cookie file, which in this case would be "c:\Program Files\Netscape\Users\John\cookies.txt". Indeed, the surreptitious harvesting of cookies files is available to those seeking it; and the privacy of Internet users are subject to invasion.

4

Another attempted solution is practiced by some industry participants. This attempted solution involves storing in persistent cookies a primary key (or database index) to a database containing data records of user information, rather than storing the private data in the persistent cookies. Thus, the unauthorized viewing or reading of a

5   primary key does not appear to be an invasion of privacy. While some, including the public, may consider such a practice as sufficient in protecting user privacy from invasion, practically it is insufficient and provides a false sense of security.

By definition, primary keys are unique within a defined universe. Thus, within a defined universe of Internet users, a single primary key uniquely identifies one or more

10   database records that relate to a specific user. Where the contents of a database are known or obtained by a party (i.e., possessed, or hacked into and harvested), an Internet user, within the defined universe, visiting a website associated with that party risks an invasion of privacy. If the user has a primary key stored in a persistent cookie on the user's hard drive, access to that cookie may allow information relating to the user in the

15   database to be referenced and used by the party to establish an undesired state between the website and the user. In addition, other information about the user that may be harvested during the visit from other cookies stored on the user's hard drive may be combined with the user's data in the database. For example, the database may only contain the user's name, address and phone number. But data harvested from the user's

20   other cookies may reveal that the user had visited a website associated with herbal treatments for those with HIV, a website associated with HIV treatment centers in the user's town and a website associated with HIV research. By combining this health-related data with the database data, the name, address and phone number of a person who appears to have HIV is now known. Where the person does in fact have HIV and sought

25   to keep his or her ailment private, this combined information results in the person's privacy being clearly invaded.

Therefore, there is a need for secure network user states.

SUMMARY OF THE INVENTION

An object of the present invention is to provide secure network user states.

Another object of the present invention is to provide secure network user states that assist in protecting user privacy.

The environment of the present invention includes at least one server and at least one computer communicatively connected together via an HTTP-based network, where user data is used to establish a state between a server and a user operating a computer.

In an exemplary aspect of the invention, a method of establishing by a server a secure state between the server and a user operating a computer, includes receiving, from the computer, a user key; creating, from the user key, a first key; encrypting, using the first key, user data; storing the encrypted user data in a cookie; naming the cookie by storing name data in the cookie; and sending the cookie to the computer; such that subsequently, a secure state between the server and the user is established by receiving the cookie and the user key from the computer; creating a second key that matches the first key; decrypting, using the second key, the encrypted user data extracted from the cookie; and establishing the secure state based on the decrypted user data.

In another exemplary aspect of the invention, a method of establishing by the server a secure state between the server and a user operating the computer, includes receiving, from the computer, a user key and a cookie having stored therein encrypted user data that may be seeded according to a format; creating, from the user key, a cryptographic key; decrypting, using the cryptographic key, the encrypted user data; and establishing the secure state between the server and the user based on the decrypted user data.

In yet another exemplary aspect of the invention, a key is created in any repeatable manner, which mathematically must include either at least one insertion or at least one deletion as defined herein.

In a further exemplary aspect of the invention, optionally, user data may be seeded according to a format before it is encrypted, such that subsequent decryption of encrypted user data will reveal whether the user data was initially seeded according to the format, which further heightens the secure nature of secure network user states.

In yet a further exemplary aspect of the invention, a computer readable medium includes instructions for establishing a secure state between the server and a user operating the computer, by causing the server to perform a method as disclosed herein.

Other features and advantages of the present invention will be apparent from the
5   accompanying drawings and the detailed description that follows.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention is illustrated by way of example and not limitation in the figures of the accompanying drawings in which like references indicate similar elements
10   and in which:

FIG. 1 shows the general environment of the present invention, in which one or more servers are connected to one or more remote computers via the Internet or other HTTP-based network.

FIG. 2a shows the sending of private data by a remote computer over the Internet
15   to a server according to the present invention.

FIG. 2b shows a form containing data fields including a field for inputting of a user key.

FIG. 2c shows a form containing a field for inputting of a user key.

FIG. 3 shows the inputting of private data and an encryption key into an
20   encryption function to produce encrypted private data.

FIG. 4a shows in a preferred embodiment a user key prior to a series of replacement steps.

FIG. 4b shows a user key after a first replacement step.

FIG. 4c shows a user key after a second replacement step.

25   FIG. 5a shows a user key prior to a series of insertion steps.

FIG. 5b shows a user key after a first insertion step.

FIG. 5c shows a user key after a second insertion step.

FIG. 6a shows a user key prior to a series of deletion steps.

FIG. 6b shows a user key after a first deletion step.

30   FIG. 6c shows a user key after a second deletion step.

FIG. 6d shows a user key after a third deletion step.

FIG. 6e shows a user key after a fourth deletion step.

FIG. 7a shows a user key prior to a series of rearrangement steps.

FIG. 7b shows a user key after a first rearrangement step.

FIG. 7c shows a user key after a second rearrangement step.

5 　 FIG. 8a shows a seed and private data in a preferred seeding embodiment prior to the combining of the seed with private data.

FIG. 8b shows a seed and private data in a preferred seeding embodiment after the seed is combined with private data.

FIG. 9 shows a cookie containing encrypted private data.

10 　 FIG. 10a shows a non-persistent cookie sent over the Internet to a remote computer that stores it in the remote computer's RAM.

FIG. 10b shows a persistent cookie sent over the Internet to a remote computer that stores it on a hard drive.

FIG. 11a shows a remote computer sending a cookie over the Internet to a first

15 server.

FIG. 11b shows a remote computer sending a cookie over the Internet to a second server.

FIG. 12 shows the decryption of encrypted private data with a user key.

20 DETAILED DESCRIPTION

Fig. 1 shows the environment of the present invention, in which one or more servers 10.1 – 10.a (for a ≥ 1) are operatively connected via the Internet 20 (or other network) to one or more remote computers 30.1 – 30.b (for b ≥ 1).

Generally, the present invention is carried out via software executing on one or

25 more servers, software executing on one or more remote computers, and user input via one or more input devices operably connected to a user's remote computer.

The private data of a user (sometimes "user data") is herein broadly defined, and may include the user's IP address, name, mailing address, email address, age, sex, credit card information, login/password combinations, preferences, hobbies, education level,

30 browsing (click) history, browsing history with click frequency, browsing preferences, assigned primary keys, assigned GUIDs, etc. In essence, private data relating to a user

includes any information that may in and of itself be personal and private, as well as information that may be personal and private when combined with other data relating to the user. Thus, any data particular to a user may be considered private data.

According to the present invention, a remote computer, associated with an

5   Internet user, comprises an operable Internet connection, Internet software, one or more computer memories for readably storing said Internet software, one or more input devices and a CPU for executing said Internet software, wherein said computer Internet connection, each of said one or more computer memories, each of said one or more input devices and said computer CPU are operatively connected to each other by at least one

10   bus. Preferably, a remote computer has at least two computer memories: RAM and a hard drive; and at least two input devices: a pointing device and a keyboard.

According to the present invention, a server comprises an operable Internet connection, one or more server memories for readably storing server software and cryptography software, and a CPU for executing said server software and said

15   cryptography software, wherein said server CPU, each of said server memories and said server Internet connection are operatively connected to each other by at least one bus. Preferably, a server has at least two server memories: RAM and a hard drive. However, since software may be stored solely in RAM, the required software may be loaded into RAM from a removable storage device (e.g., hard drive) or storage medium (e.g., diskette

20   or data cartridge), with the device or medium subsequently being removed.

The server software and cryptography software executing on a server may be implemented with any compatible programming language and/or script that functionally effectuates the present invention as claimed.

The cryptography software according to the present invention may perform

25   encryption and/or decryption. Generally, the cryptography software performs both encryption and decryption; however, where a first server only encrypts data and other related servers decrypt data, cryptography software need not perform both. The cryptography software may utilize any key-based encryption algorithm, or combination of algorithms in whole or in part, known in, taught by or obvious in light of the prior art

30   that effectuates the present invention as claimed. However, it is essential that the overall

functionality of the encryption algorithm used is one-to-one, in that the initial data results from decryption of an encrypted format of the initial data. An illustrative set of example prior art encryption algorithms and techniques from which one may draw one or more, in whole or in part, in effectuating the present invention are RSA; DSA; Diffie-Hellman;

5   Public-Key Cryptography; PGP; Signature Algorithms; DES; triple-DES; IDEA; TDEA; Blowfish; Twofish; Yarrow; Square; TEA; CAST-128; RC4; Safer SK-128; Block Ciphers, including TWOFISH; Stream Ciphers; MD2, MD4, MD5 and other techniques based on the Secure Hash Standard (SHS) or Secure Hash Algorithm (SHA-1); Digital Timestamps Supporting Digital Signatures; Secret Sharing Schemes, including Blakley's

10   Secret Sharing Scheme, Shamir's Secret Sharing Scheme and Visual Secret Sharing Schemes; Interactive Proofs; Zero-Knowledge Proofs; Message Authentication Codes; Quantum cryptography; and known or apparent variations and combinations thereof.

In a preferred embodiment, PUKALL's 128-bit stream cipher algorithm is used in effectuating the present invention, as it is known to execute with relatively great speed

15   with small text-based data; was conveniently published by ALEXANDER PUKALL in 1991 (see http://www.multimania.com/cuisinons/pc1/index.html); and by its express terms, may be used freely even for commercial applications. Use of this algorithm is highly suggested because of its fast processing speed, high encryption strength (128-bit keys) and minimal cost (free).

20   According to the present invention, associated with each user is an encryption key, which is used to encrypt private data relating to the user. An encryption key may be any number of bits consistent with the particular encryption algorithm used. However, it is important to note that the larger the key size, the stronger the encryption. The most common sizes of encryption keys are evenly divisible by eight (e.g., 40bit, 56bit, 64bit,

25   80bit, 128bit, 160bit, 256bit, 512bit, 1024bit, etc.); however other sizes may be used to the extent desirable and consistent with the particular encryption algorithm used. In a preferred embodiment, 128bit encryption keys are used. As described below, an encryption key is created by a server from a user key; therefore, an encryption key need not be, and preferably is not, archived in any computer storage device to help prevent

30   theft thereof.

USER KEY

According to the present invention, associated with each user is a user key. A user key is a value having one or more bits in length. Preferably, a user key has a number of bits evenly divisible by eight; and conveniently, ASCII characters, which represent

5   eight bits of data, may be used. In a preferred embodiment, a user key is between 48 and 256 bits in length (i.e., between 6 and 32 ASCII characters).

A user key may be archived in a memory or storage device of a server (or related server) or a user computer. Preferably, however, a user key is either memorized or otherwise recorded in a non-computer-based fashion by the user for later reference (e.g.,

10   writing it down on a piece of paper); and specifically, not archived in any computer memory or on any computer storage device.

According to the present invention, a user key may be generated by a server or selected by a user. Preferably, a user key is selected by a user, which may make the user key easier to remember by its associated user.

15   Where a user key is generated by a server, the server may either select the user key from a predefined set of possible user keys; or randomly generate the user key in any known or obvious way consistent with the present invention. The server may then send the user key to the user over the network so that the user may memorize or otherwise record the user key for later reference. Preferably, a server sends a generated key to a

20   user over a secure connection, such as SSL.

Where a user key is selected by a user, the user may input one or more bits of data in any known or apparent way. Preferably, a user may input one or more ASCII characters via an input device in any known or obvious way, such as by typing or selecting one or more characters with a keyboard or clicking one or more grid fields of a

25   grid representing ASCII characters (all ASCII character) with a pointing device (e.g., a mouse, trackball, touch pad, etc.). In a preferred embodiment, a user is sent form data by a server over the network that presents the user with a form containing a data field for input of one or more bits or characters, the form being displayed on the user's display device via browser software; and upon submission of the form, the bits or characters

30   being posted to a CGI program or script or similar program or script located on the server

via transmission back over the Internet to the server. Preferably, a user sends a user key to a server over a secure connection, such as SSL.

As shown in FIG. 2b, a user key field 73 may be displayed on a user's display device 70 for a user to enter a user key. Further, a submit button 74 may be provided for

5   submission of the form.

ENCRYPTION KEY

According to the present invention, a server may create an encryption key having e bits (e ≥ 1) from a user key having u bits (u ≥ 1) in any particular way that is

10   repeatable, in that the same encryption key will be re-created by the server or related server each time given the same user key. According to the present invention, an encryption key may be created from a user key by inserting one or more bits into the user key, deleting one or more bits from the user key, replacing one or more bits of the user key with one or more bits, and/or rearranging one or more bits of the user key. It should

15   be noted that any combination of insertions, deletions, replacements and rearrangements may be employed to the extent an encryption key of size e is created and the combination is repeatable.

In a preferred embodiment, the following program logic may be used to create an encryption key having e bits (e ≥ 1) from a user key having u bits (u ≥ 1):

20   If u = e, then replace at least one bit of the user key with its converse, resulting in u = e.
If u < e, then replace at least one bit of the user key with its converse, and then insert one or more bits into the user key until u = e.
If u > e, then delete one or more bits from the user key until u = e, and then replace at least one bit of the key with its converse, which results in u = e.

25   Optionally, rearrange two or more bits of a user key.

In a broad sense, creation according to the present invention may be effectuated solely through one or more insertions and/or one or more deletions. For example, the rearrangement and replacement of bits are combinations of at least one insertion and at least one deletion.

30   Thus, according to the present invention, insertion, deletion, replacement and rearrangement may be employed to the extent desired and regardless of the size

12

relationship between u and e, and may be employed independently or in combination; as long as ultimately, u = e and the creation process may be repeated.

The following examples are illustrative of creating an encryption key having e bits (e ≥ 1) from a user key having u bits (u ≥ 1):

5

REPLACEMENT

According to the present invention, in a replacement step, i bits (1 ≤ i ≤ u) of a user key starting from a position k (1 ≤ k ≤ [u − i + 1]) may be replaced with j bits (j ≥ 1, j ≠ i) or i bits; preferably, with i bits. For example, replacing 2 bits (i=2) from a user key

10   "011001" (u=6) starting from position 2 (k=2) with the bits "00" results in the user key "000001".

The value i may be based on a fixed value (e.g., replacing 1 bit at a time, 2 bits at a time, 3 bits at a time, etc.) or a variable value. A variable value may be generated from a formula integrated within program code (e.g., after each replacement, with i = 1

15   initially, i = i+1, i = i+3, i = ((i*13) MOD 8)+1, etc.). Alternatively, a variable value may be based on selecting from a set of predetermined values (e.g., i_set ≡ {1,2,1,3,1,2,3,1,...} ≡ 1 bit replaced, then 2 bits replaced, then 1 bit replaced, then 3 bits replaced, etc.) in any order that is repeatable or capable of repetition (e.g., sequential, reverse order, every other value, etc.).

20   In a preferred embodiment, i is based on the fixed value 8. Thus, replacing is preferably performed 8 bits at a time.

Position k may be based on a fixed value (e.g., always inserting at position k=1, k=2, ...k = u+1) or a variable value. A variable value may be generated from a formula integrated within program code (e.g., after each replacement step, with k = 1 initially;

25   k=k+1, k=k +2, k=k+3, k = ([k*13] MOD [u+1]) + 1), etc.). Alternatively, a variable value may be selected from a set of a plurality of predetermined values (e.g., k_set ≡ {1,2,1,3,1,2,3,1} ≡ in a first replacement step, i bits are replaced starting from position 1; in a second replacement step, i bits are replaced starting from position 2; in a third replacement step, i bits are replaced starting from position 1, etc.) in any sequence

30   capable of repetition. Where a value k exceeds u − i +1, k may be reduced in value such that 1 ≤ k ≤ [u − i + 1]; for example, k = (k MOD [u − i +1]) +1.

In a preferred embodiment, k is based on a variable value that is generated within program code. Specifically, the formula k = n*i+1 may be employed within a program loop, with n = 0 initially and n = n + 1 after each replacement.

The value of a bit or bits with which to replace during a replacement step may be

5   based on a fixed value (e.g., always replacing with 0's; always replacing with 1's; if replacing 1 bit, replace with "0"; if replacing 2 bits, replace with "10"; if replacing 3 bits, replace with "111"; etc.). Alternatively, the value of a bit or bits with which to replace may be based on a variable value. A variable value may be based on a formula integrated within program code (e.g., after a replacement, incrementing the value of a bit or bits to

10   be inserted by 1, with roll over); or based on selecting a value from a set of a plurality of predetermined values (e.g., v_set ≡ {1,1,0,0,0,1,0,...} ≡ in a first replacement step, if replacing 1 bit, replace with "1"; in a second replacement step, if replacing two bits, replace with "10"; in a third replacement step, if replacing 3 bits, replace with "001"; etc.) in any sequence capable of repetition. Where all the values contained in a set have

15   been selected, selection may be restarted from the first value selected.

In a preferred embodiment, the values of bits with which to replace are based on selecting variable values from a set of a plurality of ASCII characters (e.g., v_set 300 ≡ "FootBall") in any order capable of repetition. Preferably, values from v_set are selected starting from a position m, where m is based on a variable value selected from a set of a

20   plurality of predetermined values, with a predetermined value m = n*i+1 for integers n ≥ 0, in sequential order, with the order restarting from the first value after the last value is selected. For example, such a set may be m_set ≡ {17,33,9,25,41,9,1,41,1,33}. It should be noted that where a value m would require a selection of bits from v_set beyond the maximum bit position in v_set, m may be first reduced in value (e.g., m = m – 16) or

25   selection may wrap-around to the first bit position in v_set.

FIG. 4a illustrates a preferred replacement embodiment according to the present invention, in which a user key 100 may be "Goose Gossage123" having a size of 128 bits; a v_set 300 of a plurality of predetermined values may be "FootBall"; an m_set 500 of a plurality of predetermined values may be {17,33,9,25,41,9,1,41,1,33}; i having a

30   fixed value of 8; and k having a variable value of k = n*i+1, with n = 0 initially and n=n+1 after each replacement.

14

As shown in FIG. 4b, in a first replacement step, n = 0 initially, and i has a fixed value of 8. Next, a value of k is determined according to k = n*i+1, which is the value 1. Accordingly, 8 bits (i=8) may be deleted from user key 100 starting from position 1 ("G" is to be replaced), and replaced with a value from v_set 300. The first available value

5 from m_set 500 is selected, which is the value 17. Accordingly, 8 bits (i=8) starting from position 17 of v_set may be selected (bits 17 through 24), which is the value "o". Thus, the value "G" of user key 100 may be replaced with the value "o", resulting in user key 100 having the value "ooose Gossage123". The value of n may be incremented by 1 (i.e., n = 0 + 1 = 1).

10 As shown in FIG. 4c, in a second replacement step with n = 1, i still has a fixed value of 8. Next, a value of k is determined according to k = n*i+1, which is the value 9. Accordingly, 8 bits (i=8) may be deleted from user key 100 starting from position 9 ("o" is to be replaced), and replaced with a value from v_set 300. The next available value from m_set 500 is selected, which is the value 33. Accordingly, 8 bits (i=8) starting from

15 position 33 of v_set may be selected (bits 33 through 40), which is the value "B". Thus, the value "o" of user key 100 may be replaced with the value "B", resulting in user key 100 having the value "oBose Gossage123". The value of n may be incremented by 1 (i.e., n = 1 + 1 = 2).

Preferably, replacement continues for 14 more iterations ([128 ÷ i] iterations in

20 toto). As can be seen, replacing i bits of a user key with i bits from another source does not change the value of u or e.

It should be noted that in any replacement step, any set of a plurality of predetermined values may be based on one or more values contained in a user key; however, it is preferable to use values not based on a user key.

25

INSERTION

According to the present invention, during an insertion step, one or more bits may be inserted into a user key at a position k ($1 \leq k \leq e$). In a preferred embodiment, in an insertion step, i bits ($i \geq 1$) may be inserted into a user key at position k ($1 \leq k \leq [u+1]$), with the bits before insertion at positions k through u, being shifted i places to the right after insertion, and u being increased by the value i. For example, inserting the bits "01" (i=2) into a user key "011001" (u=6) at position 3 (k=3) results in the user key "01011001" with $u = u + 2$.

The value i may be based on a variable value (see below) or a fixed value (e.g., inserting 1 bit at a time, 2 bits at a time, ..., 8 bits at a time). A variable value may be generated from a formula integrated within program code (e.g., after each insertion, with i = 1 initially, i=i+1, i=i+3, i = ((i*13) MOD 7)+1, etc.). Alternatively, and preferably, a variable number may be selected from a set containing a plurality of predetermined values (e.g., i_set ≡ {1,2,1,3,1,2,3,1,...} ≡ in a first insertion, insert 1 bit; in a second insertion, insert 2 bits; in a third insertion, insert 1 bit; in a fourth insertion, insert 3 bits; etc.) in any sequence capable of repetition (e.g., sequential, reverse order, every other value, every 3 values, etc.). The insertion of a number of bits that would result in $u > e$ should be avoided, as an additional step of removing at least one bit from the user key would be required, although it is contemplated that such an insertion would be functionally compatible with the present invention and is clearly within the scope of the present invention. Thus, for example, where an insertion of i bits would result in $u > e$, i may be first reduced in value by the excess number of bits to be inserted (i.e., i = i − (i + u − e)) to avoid one or more additional steps of removing bits from a user key.

In a preferred embodiment, i is based on a variable value selected from a set containing a plurality of predetermined values, with a predetermined value = n*8, for integers $n \geq 1$ (e.g., i_set ≡ {8,16,8,32,24,8,8,32} ≡ in a first insertion, insert 8 bits; in a second insertion, insert 16 bits; in a third insertion, insert 8 bits; in a fourth insertion, insert 32 bits; etc.) in sequential order, with the order restarting from the first value after the last value is selected.

Position k may be based on a fixed value (e.g., always inserting at position k=1, k=2, ...k = u+1) or a variable value. A variable value may be generated from a formula integrated within program code (e.g., after each insertion, with k = 1 initially; k=k+1, k=k+3, k = ([k*13] MOD [u+1]) + 1), etc.). Alternatively, and preferably, a variable

5   value may be selected from a set of a plurality of predetermined values (e.g., k_set ≡ {1,2,1,3,1,2,3,1} ≡ i bits are inserted at position 1 in a first insertion step, then i bits are inserted at position 2 in a second insertion step, then i bits are inserted into position 1 in a third insertion step, etc.) in any sequence capable of repetition. Where a value k exceeds [u+1], k may be reduced in value such that 1 ≤ k ≤ [u+1]; for example, k = (k MOD

10   [u+1]) +1.

In a preferred embodiment, k is based on a variable value selected from a set of a plurality of predetermined values, with a predetermined value = n*8+1 for integers n ≥ 0, in sequential order, with the order restarting from the first value after the last value is selected. For example, such a set may be k_set ≡ {1,41,9,41,17,9,1,25,33,9}, which may

15   result in individual insertions at positions 1, 41, 9, 41, etc. Where a selected value k is greater than u + 1, k may be reduced by 8 until k = u + 1.

The value of a bit or bits to be inserted during an insertion step may be based on a fixed value (e.g., always insert 0's; always insert 1's; if inserting 1 bit, insert "0"; if inserting 2 bits, insert "10"; if inserting 3 bits, insert "111"; etc.). Alternatively, the value

20   of a bit or bits to be inserted may be based on a variable value. A variable value may be based on a formula integrated within program code (e.g., after an insertion, incrementing the value of a bit or bits to be inserted by 1, with roll over); or based on selecting a value from a set of a plurality of predetermined values (e.g., v_set ≡ {1,1,0,0,0,1,0,...} ≡ in a first insertion step, if inserting 1 bit, insert "1"; in a second insertion step, if inserting two

25   bits, insert "10"; in a third insertion step, if inserting 3 bits, insert "001"; etc.) in any sequence capable of repetition. It should be noted that the set of a plurality of predetermined values may be based on one or more values contained in a user key; however, it is preferable that such values are not based on the user key.

In a preferred embodiment, the values of bits to be inserted are based on selecting

30   a value from a set of a plurality of ASCII characters in sequential order, with the order restarting from the first value after the last value is selected. As noted above, in a

preferred embodiment inserting bits into a user key, i is based on a variable value selected from a set containing a plurality of predetermined values, with a predetermined value = $n*8+1$, for integers $n \geq 0$; and k is based on a variable value selected from a set of a plurality of predetermined values, with each predetermined value = $n*8+1$, for integers n.

5   Thus, FIG. 5a, shows a preferred embodiment of the present invention prior to an insertion of one or more bits into a user key 100 having u bits to create an encryption key having e bits, in which a user key 100 may be "Goose Gossage" such that u = 104; an i_set 200 of a plurality of predetermined values may be "8,16,8,32,24,8,8,32"; a v_set 300 of a plurality of predetermined values may be "FootBall"; and a k_set 400 of a

10   plurality of predetermined values may be "1,41,9,41,17,9,1,25,33,9".

As shown in FIG. 5b, in a first insertion, i is selected from the first available value contained in i_set 200, which is the value "8". Accordingly, 8 bits are selected starting from the first available value in v_set 300, which is the ASCII character "F". Next, the first available value from k_set 400 is selected, which is the value "1". Accordingly, the

15   8 bit character "F" is inserted into position 1 of user key 100, resulting in the value "FGoose Gossage" and u = 112.

As shown in FIG. 5c, in a second insertion, i is selected from the next available value contained in i_set 200, which is the value "16". Accordingly, 16 bits are selected from the next available value in v_set 300, which are the ASCII characters "oo". Next,

20   the next available value from k_set 400 is selected, which is the value "41". Accordingly, the characters "oo" are inserted into position 41 of user key 100, resulting in the value "FGoosooe Gossage" such that u = 128. Since after a second insertion u = e, the creation may stop.

DELETION

According to the present invention, during a deletion step, one or more bits may be deleted from a user key starting from a position k ($1 \leq k \leq u$). In a preferred embodiment, in a deletion step, i bits ($1 \leq i \leq u$) may be deleted from a user key at

5   position k ($1 \leq k \leq [u - i + 1]$), with the bits after deletion at positions k+i through u, being shifted i places to the left, and u being decreased by the value i. For example, deleting 2 bits (i=2) from a user key "011001" (u=6) starting from position 2 (k=2) may result in the user key "0001" with u = u − 2.

The value i may be based on a variable value (see below) or a fixed value (e.g.,

10   deleting 1 bit at a time, 2 bits at a time, ..., 8 bits at a time, u − e bits). A variable value may be generated from a formula integrated within program code (e.g., after each deletion, with i = 1 initially, i=i+1, i=i+3, i = ((i*13) MOD 7), etc.). Alternatively, and preferably, a variable number may be selected from a set containing a plurality of predetermined values (e.g., i_set ≡ {1,2,1,3,1,2,3,1,...} ≡ in a first deletion, delete 1 bit;

15   in a second deletion, delete 2 bits; in a third deletion, delete 1 bit; in a fourth deletion, delete 3 bits; etc.) in any sequence capable of repetition (e.g., sequential, reverse order, every other value, every 3 values, etc.). The deletion of a number of bits that would result in u < e should be avoided, as an additional step of inserting at least one bit to the user key would be required, however it is contemplated that such a deletion is

20   functionally compatible with the present invention and within the scope of the invention. Thus, for example, where a deletion of i bits would result in u < e, i may be first reduced in value by the excess number of bits; i.e., i = i − (e − [u − i]) to avoid one or more additional steps of inserting bits into a user key.

In a preferred embodiment, i is based on a variable value selected from a set

25   containing a plurality of predetermined values, with a predetermined value = n*8, for integers n ≥ 1 (e.g., i_set ≡ {8,16,8,32,24,8,8,32} ≡ in a first deletion, delete 8 bits; in a second deletion, delete 16 bits; in a third deletion, delete 8 bits; in a fourth deletion, delete 32 bits; etc.) in sequential order, with the selection restarting from the first value after the last value is selected.

Position k may be based on a fixed value (e.g., always deleting from position k=1, k=2, ..., k=u –i+1) or a variable value. A variable value may be generated from a formula integrated within program code (e.g., after each deletion step, with k = 1 initially; k=k+1, k=k +2; k=k+3, k = ([k*13] MOD [u+1]) + 1), etc.). Alternatively, and

5   preferably, a variable value may be selected from a set of a plurality of predetermined values (e.g., k_set $\equiv$ {1,2,1,3,1,2,3,1} $\equiv$ i bits are deleted starting from position 1 in a first deletion step, then i bits are deleted starting from position 2 in a second deletion step, then i bits are deleted starting from position 1 in a third deletion step, etc.) in any sequence capable of repetition. Where a value k exceeds [u – i +1], k may be reduced in

10   value such that $1 \leq k \leq [u – i + 1]$ (e.g., k = (k MOD [u – i +1]) +1).

In a preferred embodiment, k is based on a variable value selected from a set of a plurality of predetermined values, with a predetermined value = n*8+1 for integers n $\geq$ 0, in sequential order, with the order restarting from the first value after the last value is selected. For example, such a set may be k_set $\equiv$ {1,41,9,41,17,9,1,25,33,9}, which may

15   result in individual deletions starting from positions 1, 41, 9, 41, etc.

As noted above, in a preferred embodiment deleting bits from a user key, i is based on a variable value selected from a set containing a plurality of predetermined values, with a predetermined value = n*8, for integers n $\geq$ 1; and k is based on a variable value selected from a set of a plurality of predetermined values, with a predetermined

20   value = n*8+1, for integers n $\geq$ 0.

FIG. 6a, shows a preferred embodiment of the present invention prior to deletion of one or more bits from a user key 100 having u bits to create an encryption key having e bits, in which a user key 100 may be "Goose Gossage is Here" such that u=168; an i_set 200 of a plurality of predetermined values may be "8,16,8,32,24,8,8,32"; and a

25   k_set 400 of a plurality of predetermined values may be "1,41,9,41,17,9,1,25,33,9".

As shown in FIG. 6b, in a first deletion step, i is selected from the first available value contained in i_set 200, which is the value "8". Next, the first available value from k_set 400 is selected, which is the value "1". Accordingly, 8 bits are deleted from user key 100 starting from the position 1, resulting in the value "oose Gossage is Here" and u

30   = 160.

As shown in FIG. 6c, in a second deletion, i is selected from the next available value contained in i_set 200, which is the value "16". Next, the next available value from k_set 400 is selected, which is the value "41". Accordingly, 16 bits are deleted from user key 100 starting from the position 41, resulting in the value "oose ssage is Here" such that u = 144.

As shown in FIG. 6d, in a third deletion, i is selected from the next available value contained in i_set 200, which is the value "8". Next, the next available value from k_set 400 is selected, which is the value "9". Accordingly, 8 bits are deleted from user key 100 starting from the position 9, resulting in the value "ose ssage is Here" such that u = 136.

As shown in FIG. 6e, in a fourth deletion, i is selected from the next available value contained in i_set 200, which is the value "32". Since deleting 32 bits from user key 100 would result in u < e (i.e., 104 < 128), i is reduced in value by the excess number of bits; via i = i − (e− [u − i]), resulting in i = 8. Next, the next available value from k_set 400 is selected, which is the value "41". Accordingly, 8 bits are deleted from user key 100 starting from the position 41, resulting in the value "ose sage is Here" such that u = 128. Since after a fourth insertion u = e, the creation may stop.

REARRANGEMENT

As noted above, according to the present invention, one or more rearrangement steps may optionally be employed in creating an encryption key from a user key.

According to the present invention, in a rearrangement step, one or more bits of a user key may be deleted from a user key starting from position k and inserted into the user key starting from position m.

In a preferred embodiment, in a rearrangement step, i bits (1 ≤ i ≤ u) may be deleted from a user key starting from position k (1 ≤ k ≤ [u − i +1]), with the bits after deletion at positions k+i through u, being shifted i places to the left after insertion, and u being decreased by the value i. For example, deleting 2 bits (i=2) from a user key "011001" (u=6) starting from position 2 (k=2) may result in the user key "0001" with u = u − 2. Next, the one or more deleted bits may be inserted into the user key starting from position m (1 ≤ m ≤ [u +1]), with the bits before insertion from positions k through k+i

being shifted i places to the right, and u being increased in value by i. For example, inserting 2 bits (i=2) into a user key "0001" (u=6) starting from position 5 (m=5) may result in the user key "000111" with u = u + 2.

The value i may be based on a variable value (see below) or a fixed value (e.g., rearranging 1 bit at a time, 2 bits at a time, ..., 8 bits at a time, u – 1 bits). A variable value may be generated from a formula integrated within program code (e.g., after each rearrangement, with i = 1 initially, i=i+1, i=i+3, i = ((i*13) MOD 7), etc.). Alternatively, and preferably, a variable number may be selected from a set containing a plurality of predetermined values (e.g., i_set ≡ {1,2,1,3,1,2,3,1,...} ≡ in a first rearrangement, switch 1 bit with 1 bit; in a second rearrangement, switch 2 bits with 2 bits; in a third rearrangement, switch 1 bit with 1 bit; in a fourth rearrangement, switch 3 bits with 3 bits; etc.) in any sequence capable of repetition (e.g., sequential, reverse order, every other value, every 3 values, etc.).

In a preferred embodiment, i is based on a variable value selected from a set containing a plurality of predetermined values, with a predetermined value = n*8, for integers n ≥ 1 (e.g., i_set ≡ {8,16,8,32,24,8,8,32} ≡ in a first replacement, replace 8 bits with 8 bits; in a second replacement, replace 16 bits with 16 bits; in a third replacement, replace 8 bits with 8 bits; in a fourth replacement, replace 32 bits with 32 bits; etc.) in sequential order, with the selection restarting from the first value after the last value is selected.

Position k may be based on a fixed value (e.g., always replacing from position k=1, k=2, ..., or k = u–i+1) or a variable value. A variable value may be generated from a formula integrated within program code (e.g., after each rearrangement step, with k = 1 initially; k=k+1, k=k +2; k=k+3, k = ([k*13] MOD [u–i+1]) + 1), etc.). Alternatively, and preferably, a variable value may be selected from a set of a plurality of predetermined values (e.g., k_set ≡ {1,2,1,3,1,2,3,1} ≡ in a first rearrangement step, i bits starting from position 1 are rearranged with i bits starting from position m; in a second rearrangement step, i bits starting from position 2 are rearranged with i bits starting from position m; in a third rearrangement step, i bits starting from position 1 are rearranged with i bits starting from position m; etc.) in any sequence capable of repetition. Where a

value k exceeds [u − i +1], k may be reduced in value such that $1 \leq k \leq$ [u–i+ 1] (e.g., k = (k MOD [u − i +1]) +1).

In a preferred embodiment, k is based on a variable value selected from a set of a plurality of predetermined values, with a predetermined value = n*8+1 for integers n ≥ 0,

5   in sequential order, with the order restarting from the first value after the last value is selected. For example, such a set may be k_set ≡ {1,41,9,41,17,9,1,25,33,9}, which may result in individual rearrangement steps starting from positions 1, 41, 9, 41, etc.

In a rearrangement step, m may be based on a fixed value or a variable value, to the same extent as k. It should be noted that while k and m may be selected from the

10   same set of a plurality of predetermined values, preferably they are selected from different sets.

In a preferred embodiment, m is based on a variable value selected from a set of a plurality of predetermined values, with a predetermined value = n*8+1 for integers n ≥ 0, in sequential order, with the order restarting from the first value after the last value is

15   selected. For example, such a set may be m_set ≡ {17,33,9,25,41,9,1,41,1,33}, which may result in individual rearrangement steps starting from positions 17, 33, 9, 25, etc.

As noted above, in a preferred embodiment of rearranging bits in a user key, i is based on a variable value selected from a set containing a plurality of predetermined values, with a predetermined value = n*8, for integers n ≥ 1; k is based on a variable

20   value selected from a set of a plurality of predetermined values, with a predetermined value = n*8+1, for integers n ≥ 0; and m is based on a variable value selected from a set of a plurality of predetermined values, with a predetermined value = n*8+1, for integers n ≥ 0.

FIG. 7a shows a preferred embodiment of rearrangement according to the present

25   invention prior to rearrangement of one or more bits in a user key 100; in which a user key 100 may be "Goose Gossage123", with u = 128; an i_set 200 of a plurality of predetermined values may be "8,16,8,32,24,8,8,32"; a k_set 400 of a plurality of predetermined values may be "1,41,9,41,17,9,1,25,33,9"; and an m_set 500 of a plurality of predetermined values may be "17,33,9,25,41,9,1,41,1,33".

As shown in FIG. 7b, in a first rearrangement step, i is selected from the first available value contained in i_set 200, which is the value "8". Next, the first available value from k_set 300 is selected, which is the value "1". Accordingly, 8 bits may be deleted from user key 100 starting from position 1 ("G" is deleted), resulting in user key 100 having the value "oose Gossage123" and u =120. Next, the first available value from m_set 500 is selected, which is the value "17". Accordingly, "G" may be inserted into user key 100 starting from position 17, resulting in user key 100 having the value "ooGse Gossage123" and u = 128.

As shown in FIG. 7c, in a second rearrangement step, i is selected from the next available value contained in i_set 200, which is the value "16". Next, the next available value from k_set 300 is selected, which is the value "41". Accordingly, 16 bits may be deleted from user key 100 starting from position 41 (" G" is deleted), resulting in user key 100 having the value "ooGseossage123" and u = 112. Next, the next available value from m_set 500 is selected, which is the value "33". Accordingly, " G" may be inserted into user key 100 starting from position 33, resulting in user key 100 having the value "ooGs Geossage123" and u = 128.

Rearrangement, as shown, may continues for as many iterations as desired. As noted, rearrangement is optional. As can be seen, replacing i bits of a user key with i bits from another source does not change the value of u or e.

SEEDING

According to the present invention, seeding may be optionally employed, and when employed, is part of the cryptography software. A seed may be combined with private data prior to encryption according to a particular format (e.g., intermixed symmetrically or asymmetrically with the private data, appended to the head or tail of the private data, etc.). After decryption of encrypted private data that was seeded, the seed may be extracted and compared to its known value. Where the extracted value (potential seed) deviates from the known seed value, a server may know that either the encryption was unsuccessful or the encrypted private data was not encrypted by the server.

A seed may be one or more bits. Preferably, a seed is 40 bits in length.

The value of a seed may be predetermined (e.g., "Beer!") or generated based on one or more bits in the private data (e.g., checksum, parity, etc.). Preferably however, a seed has a predetermined value.

FIG. 8 illustrates a preferred seeding technique according to the present invention. As shown in FIG. 8, a seed 60 may have the value "Beer!" and private data 50 may have the value "Gossage". As shown in FIG. 8b, seed 60 may be concatenated to the head of private data 50 resulting in modified private data 50 "Beer!Gossage". Thus, as seeded, private data 50 may be encrypted. Upon decryption, 40 bits may be extracted from the head of private data 50 and compared to determine whether it is seed 60.

The present invention is now described in operation:

As shown in Fig. 2a, a user's computer may contact a server according to the present invention by sending a request for data, which contains private data 50, via the user's computer 30.j over the Internet 20 to a server 10.i. After receiving the private data 50, a server 10.i causes the data 50 to be stored in a memory 11.i (preferably RAM) for subsequent encryption. The user may contact a server by entering the URL or Internet address of the server or by clicking or selecting a bookmark or hyperlink directed to the server. Subsequently, the user may continue to enter private data 50 relating to the user, which may include any private data as previously defined. As shown in FIG. 2b, this is preferably performed by a user filling out a form, via keyboard entry, displayed on a display device 70 via browser software. Referring again to Fig. 2b, a form may contain a name field 71 and an email field 72 for inputting of a name and an email address, respectively; and further, a form may contain a submit button 74 for submission of a completed form. Upon submission of the form, the data may be posted to a CGI Program or script or similar program or script located on the server via transmission over the Internet to the server.

It should be noted that the clicking history of a user, as that user peruses web pages, banners and/or links may also be considered private data. Thus, the submission of data may occur over one or more transactions and need not be limited to submitted form data. Further, as expressly defined above, private data may be a primary key; thus, the sending of private data by a user is not necessarily required.

The private data sent by the user, in whole or in part, represents a state between the user and the server and thereby dictates in whole or in part the private data to be stored in a cookie by the server. For example, a user may send his last name, which may be "Gossage". Preferably, as shown in Fig. 2a, the private data 50 may be transferred

5   between the user's computer 30.j and server 10.i via a secured connection 15, such as SSL.

It should be noted that some precoding and decoding may be required with the particular encryption algorithm employed. For example, the private data may be concatenated in a particular order to form a string having fixed or variable length fields.

10   Another form of coding is binary/text conversion, and may arise, for example, where a particular encryption algorithm/technique requires input data in binary format and the input data is initially in text format; and may also be optionally employed after output. Any binary/text conversion and/or coding may be optionally employed prior to input, as well as subsequent to output, as long as one-to-one correspondence between the precoded

15   and decoded data is preserved.

As shown in Fig. 3, encryption of private data 50 is effectuated by inputting the private data 50 and an encryption key 101.k (for $k \geq 1$) into encryption function 100, which produces encrypted private data 110. It should be noted that the format of the particular key used will be dictated by the particular cryptography algorithm employed.

20   For example, the PUKALL algorithm is designed to accept 16-character strings as keys.

According to a preferred embodiment, the PUKALL encryption algorithm accepts a 16-character string as a 256-bit key, such as the 16-character string "Remsaalps!123456". Encrypting the value "Gossage" with this key via the PUKALL algorithm may produce the value "¾M}7¿□c÷□".

25   There are six parameters that may be assigned to a cookie: (1) the name of the cookie, (2) the value of the cookie, (3) the expiration date of the cookie, (4) the path the cookie is valid for, (5) the domain the cookie is valid for, and (6) a flag representing the need for a secure connection to exist to use the cookie.

According to the present invention, the first two parameters must be explicitly

30   assigned values: (1) the name of the cookie, and (2) the value of the cookie. The next four parameters may be optionally explicitly assigned values: (3) the expiration date of

the cookie, (4) the path the cookie is valid for, (5) the domain the cookie is valid for, and (6) a flag representing the need for a secure connection to exist to use the cookie. These optional parameters may be explicitly assigned values to improve security and/or functionality, and are discussed in further detail herein.

5   Generally, creating a cookie involves replicating the HTTP cookie header in some fashion so that browser software executing on a remote computer will recognize and store the cookie.

According to the present invention, name data is assigned to the name field of a cookie, such as the name "ywi". Further, the encrypted private data with the embedded 10   key is assigned to the value field of a cookie.

The optional parameter (3), the expiration date of the cookie, may be assigned to the expires field of a cookie to direct browser software executing on a remote computer whether to store the cookie on a storage device, e.g., a hard drive. If not explicitly assigned a value, the expires field defaults to end-of-session and the browser preserves 15   the cookie only in memory (RAM) until the browser session is closed. Such a value may be a past date, or a future date such as "Mon, 09-Dec-2002 13:46:00 GMT", which if processed prior to expiration, may be stored in a storage device. A resulting HTTP header representing this cookie may be as follows:

        Content-type: text/html
20   Set-Cookie: ywi="¾M}7¿□c÷□"; path=/;
                     expires = Mon, 09-Dec-2002 13:46:00 GMT

Upon receiving this header, browser software executing on a remote computer may store the cookie to a storage device.

The optional parameter (4), the path the cookie is valid for, may be explicitly 25   assigned a value, such as "/computerstore". This causes to be set the URL path the cookie is valid within. Thus, pages outside the path "/computerstore" cannot read or use the cookie having this value. Explicitly assigning a value to this parameter would be advantageous where multiple websites exist within a domain, such as www.thissite.com and www.thissite.com/otherparty, and sharing of cookies between the servers associated 30   therewith is undesired. If not specified, the value defaults to the path of the document creating the cookie.

The optional parameter (5), the domain the cookie is valid for, may be explicitly assigned a value, such as ".thissite.com". Where a website uses multiple servers within a domain, it may be desirable to make the cookie accessible to pages on any of those servers. Thus, a cookie may be assigned to an individual server or to an entire Internet

5   domain. Here, all servers within the domain www.thissite.com may access the cookie so defined. The default value if not explicitly set is the full domain of the document creating the cookie.

The optional parameter (6), a flag representing the need for a secure connection to exist to use the cookie, should only be used under a secure sever condition, such as SSL.

10   Where secure transactions are implemented, this parameter heightens security between a server and a remote computer. If not explicitly set to TRUE, this defaults to FALSE.

In sending the cookie to a remote computer, the server effectuates the creation of an HTTP header which is sent along with a requested page. This causes the value of the cookie to be sent to a remote computer, received thereby, and in conjunction with

15   browser software executing thereon, stored in memory or on a storage device, such as a hard drive.

As shown in Fig. 9, according to the present invention, a cookie 300 has at a minimum a name field 301 with a value representing the name of the cookie, such as "ywi". Further, the cookie 300 has a value field 302 containing the encrypted private

20   data 110, which may be "¾M}7¿□c÷□".

It is important to note that cookies, as they are implemented today, are transferred between a server and a browser as an HTTP header and the specifications for this header are explicitly set forth in RFC 2109, which has been readily and freely available over the Internet. Further, the setting and reading of cookies can be effectuated with a plurality of

25   languages and/or scripts, and the particular choice of language and/or script is not important to the present invention insofar as it does not deviate from the teachings of the invention. Examples of languages and scripts are as follows: JavaScript, PERL, LiveWire, ASP, Virtual Basic ("VB") and VBScript. Further, custom software may be used, via C, C++, etc., to the extent that a compatible HTTP header is created.

As shown in Figs. 10a and 10b, a cookie 300 is sent over the Internet 20 to a user's remote computer 30.j, which stores the cookie 300 in a read-write device, which may be the remote computer's RAM 31.j or storage device, such as a hard drive 32.j.

When a user subsequently requests a page from the server or a related server, i.e.,

5   a request from the browser to a server, the cookie header is modified slightly from that which created the cookie. For example, the header may be as follows:

Content-type: text/html
Set-Cookie: ywi = "¾M}7¿□c÷□"

Here, a server is made aware of the cookie named "ywi" having the value of the

10   encrypted private data with embedded key. Generally, retrieving a cookie from a header does not require actual reading of the HTTP Cookie: header, since most languages automatically read this header for the programmer and make it accessible through a programming variable or object. As with creating a cookie, accessing a cookie may be effectuated with a plurality of languages and/or scripts, and the particular choice of

15   language and/or script is not important to the present invention insofar as it does not deviate from the teachings of the present invention. Examples of languages and scripts are discussed above.

As shown in Fig. 11a, a user's remote computer 30.j may send a cookie 300 back to server 10.i that originally sent the cookie 30.j to the user's computer 30.j for storage of

20   the cookie 30.j in server memory 11.i for subsequent processing.

As shown in Fig. 11b, a user's remote computer 30.j may send a cookie 300 to a server 10.2, which is not the server that originally sent the cookie 300 to the user's computer 30.j. As also shown Fig. 6b, server 10.2 stores cookie 30.j in server memory 11.2 for subsequent processing.

25   Once the value of the cookie is accessed by a server, the encrypted private data may be extracted from the data field of the cookie.

As shown in FIG. 2c, a user may again be presented with a form having a user key data field 73 for inputting and submission to a server of the user's associated user key, which is required for recreation of the user's associated encryption key.

As shown in FIG. 12, upon submission of the correct user key, a server may then recreate the encryption key 101.k used to encrypt the private data according to the same creation process used to create the encryption key initially. Referring again to FIG. 12, a server may then decrypt encrypted private data 110 by inputting encrypted private data

5   110 and the recreated encryption key 101.k into decryption software 106, which results in decrypted private data 210.

With this decrypted private data, a state may be created between the server and the remote computer of a user. For example, the decrypted data may be the user's last name, such as "Gossage" wherein the server may now know the identity of the user; thus

10   creating a state between the server and the user.

As can be seen, the present invention efficiently protects the privacy of Internet users by protecting private user data available for establishing Internet user states.

In the foregoing specification, the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various

15   modifications and/or changes may be made thereto without departing from the broader spirit and scope of the invention. For example, where the size of a user key, as submitted by a user, is equal to the size required of an encryption key, the user key may be used directly as an encryption key. Accordingly, the specification and drawings are to be regarded in an illustrative and enabling rather than a restrictive sense.

20